Stabilizing NeRF Training with Batch Normalization

Joshua Ahn

December 8, 2023

1 Introduction

A now seminal work in 3D Reconstruction, Neural Radiance Fields (NeRFs)[1] are the current state-of-the-art approach to the problem of inverse computer graphics; that is, reconstructing complex 3D scenes given only 2D images of said scene and their known camera poses. To do so, they use the volumetric rendering equation[2], which reasons about transmittance (the probability that a ray of light travels from point A to point B) in a scene. The effect is that solid object surfaces have very high opacity values (and low transmittance) while points in empty space have little to no opacity (and thus have very high transmittance).

In this work, I am interested in analyzing the original NeRF paper, which uses an 8-layer MLP to encode geometric information about the scene (in this case, color and volume density). The MLP utilizes the ReLU activation function in between each layer but interestingly, it initializes the linear layers using the standard normal distribution. This initialization scheme does not take into account the *gain* of the ReLU activation function, which has the effect of causing the MLP to output values on a distribution with 0 mean and 0 variance at initialization[3]. This is bad practice according to conventional deep learning practices as it is has been established through decades of work that ensuring well-shaped distributions of both the activation values and their gradients (i.e., 0 mean and unit variance) is critical to convergence[4][5].

In line with this wisdom, I find that the MLP easily converges to local minima in the optimization landscape, which leads to worse rendering quality and poor performance at inference time. In fact, the only reason why NeRF is able to converge at all with this poor initialization scheme is due to the superpowers of the Adam optimizer[6]; I tried 8 experiments training NeRF with SGDM[7] and observed complete failures on all 8 trials.

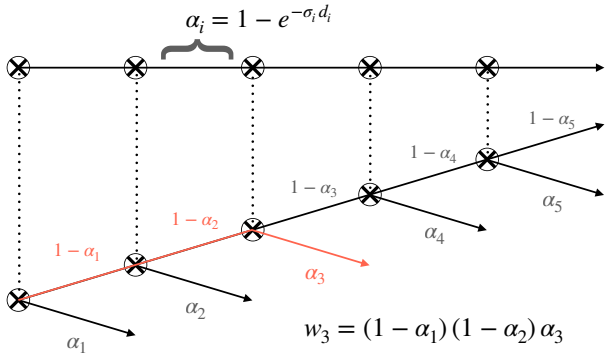

Figure 1: A discretized view of volume rendering. **Top**: a ray is cut into intervals, each with a density $\sigma_i \geq 0$ and interval length d_i. **Bottom:** illustration of the weight given to 3rd interval.

This work seeks to address this obvious deficiency of NeRF by using batch normalization [8] to assist with *stabilizing* training and ensuring that the model is able to reliably converge across all test runs. I try three different variants of the batch norm module, two of which relies on a priori knowledge distilled from the volume rendering equation itself. I find that the batch norm modules have varying degrees of success at inference time, although all modules demonstrate reliable convergence across multiple experiments.

2 Method

Volume Rendering NeRF optimizes a neural mapping from a spatial location z to color and volumetric density f_σ. There are many choices for this mapping, including MLPs[9][10], voxel grids[11][12][13], and hybrid hashgrid-MLP encodings [14], but the core idea of volumetric rendering remains a constant in all NeRF architectures. Auxiliary inputs are also sent as encoded features into this mapping, such as viewing direction[1], time[15], scene-specific embeddings[16], geometric priors[17], diffusion model priors[18][19], and more.

To render an image, we integrate $(\mathbf{c}, \sigma)$ over points on each ray $\mathbf{z}(t) = \mathbf{o} + t\mathbf{d}$ by

$$\int_0^\infty w(t) \cdot \mathbf{c}(t)\, dt = \int_0^\infty \underbrace{\sigma(t)\, T(t)}_{=\partial_t(1-T(t))} \cdot \mathbf{c}(t)\, dt, \quad (1)$$

where the transmittance $T(t)$ is the probability that light traverses the interval $[0, t]$ without dying off.

In practice, the ray is discretized into segments, each with length d_i. Assuming constant volume density and color within the segment, volume rendering takes on the form of alpha compositing, where the "over" operation [20] is applied in a back-to-front order. See Fig. 1 for a tree-branching analogy.

$$\alpha_i = 1 - e^{-\sigma_i d_i} \quad \text{with} \quad w_i = \prod_{j<i} (1 - \alpha_j) \cdot \alpha_i. \quad (2)$$

The final color is produced by the expectation w.r.t. the probability mass function $\sum_i w_i\, \mathbf{c}_i$.

Now, using (2) and simple algebra, we can solve for σ:

$$\sigma(\alpha, d) = -\frac{1}{d} \log(1 - \alpha). \quad (3)$$

Note that for solid surfaces, α, which measures opacity, approaches 1 while for empty scenes, α approaches 0. Using this insight and some example interval lengths d, which is on the numerical range of $[0.001, 0.3]$ for the scenes we consider, we compute the corresponding σ values in Tab. 1. We observe that for high α values (which represent object surfaces), NeRF must learn very large σ values that approach the numerical range of hundreds or even thousands. This is a huge challenge for an MLP that A) is poorly initialized and B) does not normalize the activation values between layers as the model needs to learn huge values which can easily cause training instability. I refer the reader to several Github issues where other members of the community have run into similar training instability issues.[1][2]

	$d = 0.005$	$d = 0.010$	$d = 0.015$
$\alpha = 0.01$	2.0	1.0	0.7
$\alpha = 0.50$	138.6	69.3	46.2
$\alpha = 0.75$	277.3	138.6	92.4
$\alpha = 0.99$	921.0	460.5	307.0

Table 1: Example desired σ values for various alpha values and interval lengths using $\sigma = -\frac{1}{d}\log(1-\alpha)$. As the desired alpha value increases, the required value for σ across all interval lengths sharply increases.

Batch Norm to the Rescue To rectify this, we propose to use batch normalization as a regularizer on the learnable weight matrices in each layer of the MLP. As a recap, batch norm normalizes an input distribution to have mean 0 and unit variance. It then applies an affine transformation $f(x) = \gamma \cdot x + \beta$, where γ and β are both learnable parameters. However, at initialization, f is simply the identity transform as γ and β are initialized to be 1_V and 0_V respectively.

In my experiments, I try adding three variants of batch normalization to the NeRF model. First is the original batch norm implementation (which we denote as *standard BN*). The second and third variants employ Tab. 1 as a heuristic and apply a log-linear scaling on γ for successive layers. As we know that

[1] https://github.com/yenchenlin/nerf-pytorch/issues/38

[2] https://github.com/yenchenlin/nerf-pytorch/issues/82

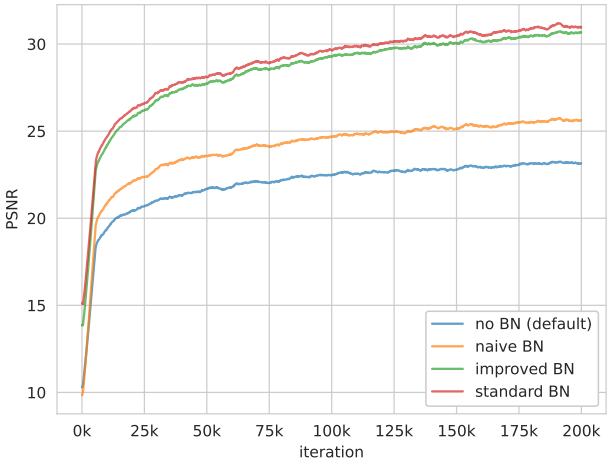

Figure 2: Average convergence curves for the 4 NeRF models on all 8 blender scenes. All 3 batch norm modules beat the default NeRF baseline but standard BN provides the best convergence.

σ needs to be very large for points on object surfaces, we can guide the model to more easily predict these large values by encouraging the distribution of the MLP's output to have mean 0 with a very large variance. The idea is that centering the distribution around 0 ensures that the vast majority of spatial locations z are mapped to $\sigma = 0$ (and thus have a corresponding local alpha value of 0) which is correct as a vast majority of the scene is empty. The large variance however ensures that a small number of those spatial locations (particularly those on solid object surfaces) get mapped to extremely large σ values, resulting in high local alpha values and thus correctly capturing scene geometry.

To accomplish this, I change the γ values directly. For the second batch norm variant, which I denote as *naive BN*, I set $\gamma_i = 2^i$ for $i = \{0, 1, ..., 6, 7\}$. γ_1 remains unchanged, while γ_8 is initialized to be 128_V. In the third variant, denoted as *improved BN*, I apply a less aggressive initialization scheme and set $\gamma_i = 2^i$ for $i = \{-2, -1, ..., 4, 5\}$ such that $\gamma_1 = 0.25_V$ while $\gamma_8 = 32_V$. Note that the activation function for all layers is ReLU, which results in an exponential distribution as all negative values are clipped to 0 in the forward pass.

3 Experiments

I train the four variants of NeRF under consideration (no BN, standard BN, naive BN, improved BN) on all 8 scenes from the Blender synthetic dataset for a total of 32 experiments. In line with previous work, we adopt the conventional recipe of Linear $\rightarrow$ BN $\rightarrow$ ReLU[21]. Note that these experiments are trained on nerf-pytorch[22], a PyTorch implementation that has been numerically tested and matches the original Tensorflow implementation.

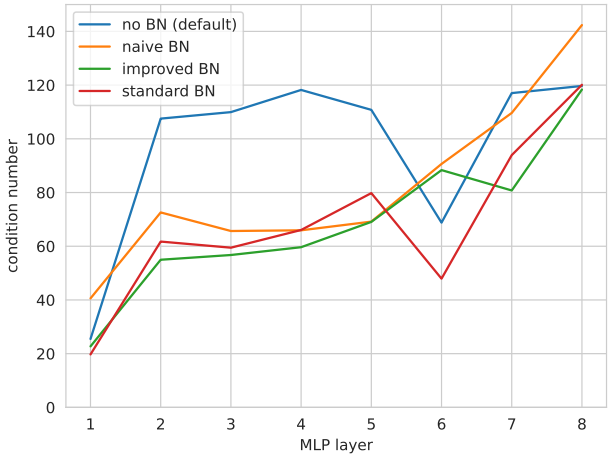

Figure 3: Average condition numbers for the weight matrices of each layer of each of the 4 NeRF MLPs across the 8 scenes from the blender dataset.

As the output of the volume rendering process is a rendered image, the loss function is simply the mean-squared error (MSE) between the pixels of the rendered image and the pixels of the ground truth image for some given camera pose. The MSE loss can be converted to a metric called Peak Signal-to-Noise Ratio (PSNR), which compares how 'similar' two images are to each other. PSNR is measured in log space and higher values corresponds to more accurate renderings.

I provide PSNR results of all 32 NeRF experiments on the evaluation set for the given scene, as well as analyze intermediate activation distributions, output σ distributions, and provide spectral analyses of the weight matri-

	chair	drums	ficus	hotdog	lego	materials	mic	ship	avg
no BN (default)	14.04	**22.91**	**28.91**	**35.06**	**30.98**	8.74	13.04	**27.28**	22.61
expected performance	32.00	24.01	29.13	35.18	31.54	28.62	31.91	27.65	30.13
standard BN	25.22	21.15	24.59	29.76	24.97	20.62	25.73	21.35	24.17
naive BN	23.70	22.19	14.23	24.56	27.08	23.53	13.04	24.51	21.61
improved BN	**29.54**	22.40	26.56	31.21	27.62	**24.53**	**29.26**	25.01	**27.02**

Table 2: Evaluation PSNR ↑ values of NeRF on all 8 scenes from the blender synthetic dataset[1]. 'Expected' lists the performance we expect from the default model but the optimization can fail randomly (as marked with red) or converge to a suboptimal location in the parameter space (as marked with orange). Note that a 1 or 2 point drop in PSNR rendering quality is significant as PSNR is calculated in log space.

ces to determine if our proposed method does indeed assist with reliable convergence.

General Results Tab. 2 shows the advantage of our improved BN module over the other models. The default model should match the values of the expected performance, but as discussed earlier, suffers from instability during training causing the optimization to fail randomly. Batch norm, in the three forms we analyze here, ensures consistent rendering quality but only the improved BN module is able to approach the expected performance baseline.

In Fig. 2, we show that the standard BN and improved BN networks have the best convergence curves across all 8 scenes. It is interesting to consider that the standard BN model has such good convergence but does poorly at inference time, as demonstrated in Tab. 2; this is a classic case of overfitting and we suspect that BN's inductive bias of normalizing all activation distributions to match a standard Gaussian prevents NeRF from effectively learning large σ values for spatial locations unseen during training. The naive BN module suffers from poor convergence as the initialization scheme on γ is too aggressive, while the default model suffers from instability during training.

I also analyze the SVD of each of the layers in the MLP. Fig. 3 shows the condition numbers (the ratio of the largest singular value to the smallest one) which is a good statistical measure for the amount of instability in

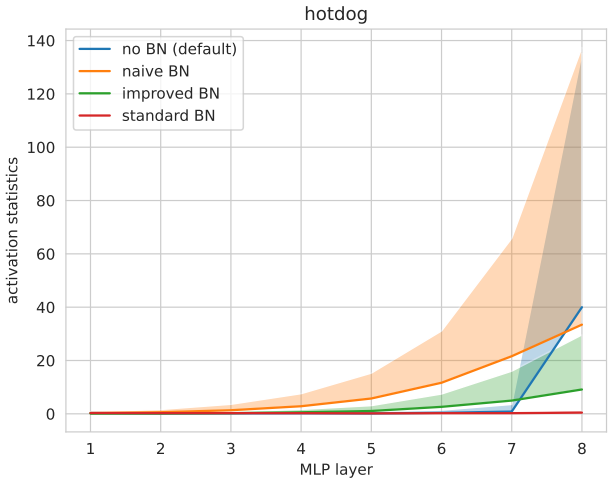

Figure 4: Intermediate distributions for each layer in the MLP for all 4 NeRF models. The solid line represents the mean, while the shaded region represents the standard deviation. Here we only plot $\mu + \sigma$ (and not $\mu - \sigma$) for ease of visibility.

the network. If the condition number ω is close to 1, then the model will be robust to changes in the input, but as ω gets larger, the model will become less and less robust to small changes in the input, demonstrating instability. We find that, interestingly, using batch normalization does not assist in regularizing the weight matrices of the linear layers as the figure shows that all the layers across all the different types of MLPs have very large condition numbers. Instead, we will see that batch norm only smooths the distribution of activation values between each layer, as opposed to 'smoothing' out the layers themselves.

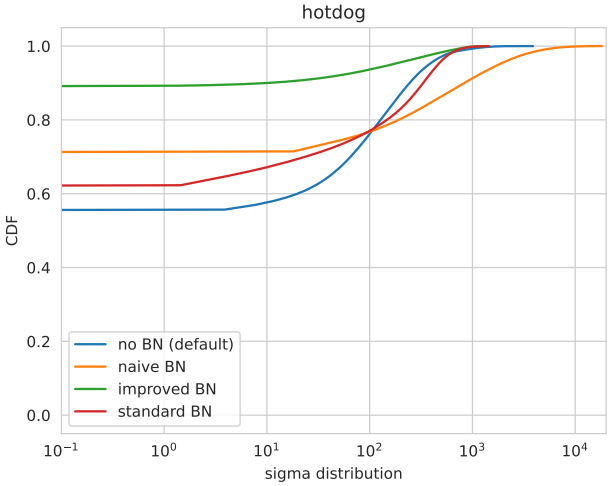

Figure 5: The CDF of the output σ distributions for the 4 NeRF modules on the hotdog scene. Because this is an approximation of the true CDF (i.e., I compute a histogram with 1000 bins), there are angular 'bumps' in this plot.

Hotdog Analysis We now take a deeper look at the hotdog scene, where the default model was able to converge very well.

In Fig. 4, we see that the naive BN and improved modules follow the expected pattern - there is a smooth upwards exponential transition of the distribution, which reflects the initialized γ values in the BN modules. The standard BN module is a flat line because all intermediate distributions are normalized to mean 0 and variance 1. The default model is very interesting as the distributions from the first 7 layers are "well-behaved"; it is only at the 8th layer that we observe a huge shift in the distribution, which reflects the necessity of the model to learn large σ values for spatial locations along solid object surfaces. However, this giant shift in the distribution is very bad (an example of a very large internal covariate shift), and a very good indicator of why the default NeRF MLP is so unstable during training.

In Fig. 5, we query a uniformly dense grid of 200^3 points for their σ values and visualize their Cumulative Distribution Function (CDF). There is a straight line from $\sigma = 10^0$ to the left because the vast majority of spatial locations in the scene are empty and have an associated $\sigma = 0$. While all 4 NeRF models are able to learn large σ values on the numerical range of hundreds and even thousands, the naive BN module learns σ values greater than 10000, which is too extreme. This is directly caused by the very aggressive initialization scheme, and the result is that the model converges poorly. Our less aggressive improved BN module learns σ values which are capped at 1000, which is in line with our a priori reasoning.

4 Conclusion

NeRF is a state-of-the-art approach to the problem of inverse computer graphics. However, the original architecture is prone to poor convergence, due in large part to the requirement of the model to learn very large σ values and to the poor initialization scheme of the MLP layers. We propose three variants of batch norm, all of which improve stability during training (none suffer from random failure modes). However, we observe that using a priori knowledge about the volume rendering equation can guide our initialization scheme of the batch norm modules themselves, which we observe helps boost performance.

References

[1] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *Eur. Conf. Comput. Vis.*, pages 405–421, 2020.

[2] James T. Kajiya. The rendering equation. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '86, page 143–150, New York, NY, USA, 1986. Association for Computing Machinery.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015.

[4] Christopher M Bishop. *Neural networks for pattern recognition*. Oxford university press, 1995.

[5] Lei Huang, Jie Qin, Yi Zhou, Fan Zhu, Li Liu, and Ling Shao. Normalization techniques in training dnns: Methodology, analysis and application. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.

[6] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[7] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1139–1147, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[8] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.

[9] Jonathan T Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P Srinivasan. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5855–5864, 2021.

[10] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5470–5479, 2022.

[11] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. Tensorf: Tensorial radiance fields. In *Eur. Conf. Comput. Vis.*, pages 333–350, 2022.

[12] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2022.

[13] Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 5501–5510, 2022.

[14] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics (ToG)*, 41(4):1–15, 2022.

[15] Albert Pumarola, Enric Corona, Gerard Pons-Moll, and Francesc Moreno-Noguer. D-nerf: Neural radiance fields for dynamic scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10318–10327, 2021.

[16] Ricardo Martin-Brualla, Noha Radwan, Mehdi SM Sajjadi, Jonathan T Barron, Alexey Dosovitskiy, and Daniel Duckworth. Nerf in the wild: Neural radiance fields for unconstrained photo collections. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2021.

[17] Sen Wang, Wei Zhang, Stefano Gasperini, Shun-Cheng Wu, and Nassir Navab. Voxnerf: Bridging voxel representation and neural radiance fields for enhanced indoor view synthesis. *arXiv preprint arXiv:2311.05289*, 2023.

[18] Rundi Wu, Ben Mildenhall, Philipp Henzler, Keunhong Park, Ruiqi Gao, Daniel Watson, Pratul P. Srinivasan, Dor Verbin, Jonathan T. Barron, Ben Poole, and Aleksander Holynski. Reconfusion: 3d reconstruction with diffusion priors. *arXiv*, 2023.

[19] Kyle Sargent, Zizhang Li, Tanmay Shah, Charles Herrmann, Hong-Xing Yu, Yunzhi Zhang, Eric Ryan Chan, Dmitry Lagun, Li Fei-Fei, Deqing Sun, and Jiajun Wu. ZeroNVS: Zero-shot 360-degree view synthesis from a single real image. *arXiv preprint arXiv:2310.17994*, 2023.

[20] Thomas Porter and Tom Duff. Compositing digital images. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 253–259, 1984.

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[22] Lin Yen-Chen. Nerf-pytorch. `https://github.com/yenchenlin/nerf-pytorch/`, 2020.